UBER Analysis and Self Driving Implementation (DC4100)

Oliver Matthew Bowker (220263618)

June 1, 2022

Contents

1 Introduction

Uber is ride-sharing company that was first created in March 2009 [1] and has since become one of the most well known companies in the world. Uber describes itself as being:

> *'a tech company that connects the physical and digital worlds to help make movement happen at the tap of a button.'* [2]

Alongside it's ride-sharing apps, for both riders and drivers, it also has Uber Eats which is a takeaway delivery service, providing apps for restaurants, deliverers and customers. However for this report I will only be focussing on the ride-sharing aspect of the company.

Within this report I aim to create documentation that represents Uber from both an application and business level using tools such as use case diagrams, sequence diagrams, Archimate and more. This report was written with some insights from this **case study**.

Tools used - To create my diagrams I have solely used **draw.io**. The creation of the report was done using LaTeX and is stored in this **github repository**.

2 Requirements and Use Cases

In this section of the report I will create a use case diagram containing the relevant actors of th system and break down why they are there. After I will take one of these use cases and expand on it in a use case description. Finally I will list some of the user requirements and functional and non-functional system requirements.

2.1 Use Cases

The first thing to do is identify all the potential stakeholders for the project. Below is a viewpoint hierarchy I created to represent these stakeholder.

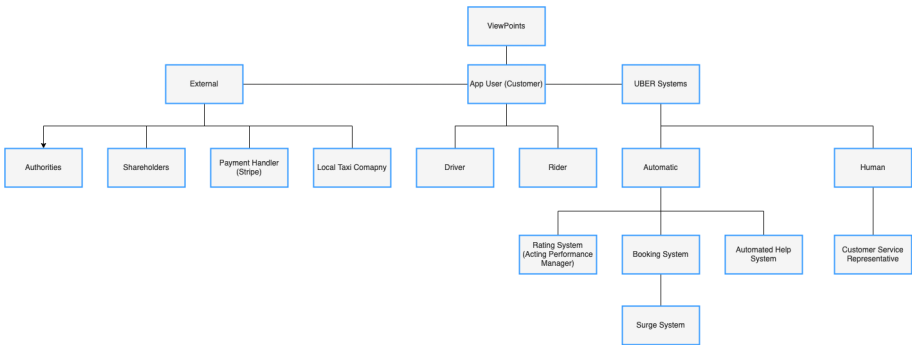

Figure 1: Viewpoint hierarchy for Uber app.

From this I was then able to 'pull' some of these stakeholders out of this viewpoint hierarchy and create my use case diagram, which can be seen in full at **Appendix A**.

2.1.1 Driver/Rider

The biggest actor in the Uber ecosystem has to be the customer. Within the case study for this report the authors claim:

> *'drivers are "end-users" of the application'*

whilst citing Ubers' contract to UK drivers. For this reason I have grouped them closely here.

Figure 2: Figure showing the rider and driver in the use case diagram.

Both **rider** and **driver** actors share a few use cases. Both can create accounts, request emergency assistance during the ride and, optionally, cancel a requested/accepted ride. They're is also a link, in functionality not in use case, between 'Confirm Ride Completion' and 'Receive Ride End Notification' as the driver triggers the latter for the user. This will be discussed more later in the report.

For the rider I have two separate use cases 'Request Ride' and 'Schedule Ride' which seem extremely similar. The 'Request Ride' use case is solely referring to a rider getting a ride now, whereas scheduled is at a later date. I believe as these use cases are documented at a lower level larger differences in functionality will appear. For this reason I have put them as separate use cases.

2.1.2 Payment Processor

The **payment processor** actor in the diagram represents an external third party that is used to handle monetary transactions in the Uber ecosystem. In a real world application this could be a company such as Stripe [3]. The use cases are mostly simple, with the main functionality handling the processing of ride payments and tips, the only bit of other functionality is the validation of the payment, which can fail and therefore needs to send back a validation error in this case.

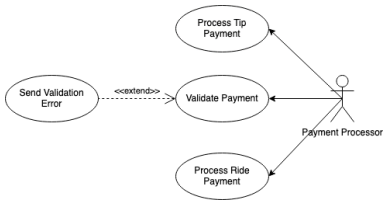

Figure 3: Figure showing the payment processor actor in the use case diagram.

2.1.3 Other Actors

There are some smaller actors that play a role in the system. The first being a **customer service representative**, which is responsible for accepting user complaints/issues. I have included an optional/extend 'Direct to Automated System' use case as in the **case study** it states that when drivers make complaints they *'believe that software is creating initial responses based on the keywords in their text'*.

Figure 4: Figure showing the CSR actor in the use case diagram.

The final actor is the **Police/Authorities**. I have included this actor as Uber offers a function to both rider and river that allows them to contact the emergency services in case something is going or has gone wrong with their current ride.

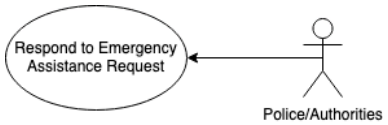

Figure 5: Figure showing the Police/Authorities actor in the use case diagram.

Use Case Number: 1 **Use Case Name:** Receive Ride End Notification
Goal: To notify the rider once their ride has concluded.
Brief Description: This use case happens when a driver has completed the drop off of a rider to their specified location. This notification leads the user to being able to leave a tip and a rating for their ride. Although the primary path does not contain both of these, the ideal situation is for the rider to leave a review for their driver.
Actors: The Driver as they have to confirm the completion for the notification to be sent. The Rider as they are the recipient of the notification.
Frequency of Execution: Hundreds of thousands of rides a day are completed using Uber.
Scalability: Uber must be able to support millions of requests for this use case on special occasions/events.
Criticality: This is an extremely important feature due to rider being able to rate the driver at this point, which allows Uber to manage the quality of their drivers through their rating system.
Other non-functional Requirements: None identified.
Preconditions: The rider must have already requested and been accepted for a ride and arrived at their destination. The driver of that ride must have also confirmed the drop off. Both rider and driver must also already have signed up and be able to use all functionality the app offers.
Postconditions: None identified.
Primary Path: 1. Customer is dropped off at their specified location. 2. The customer receives a notification that their ride has ended. 3. The customer clicks on that notification and is taken to the review ride/driver screen. 4. The customer leaves a review of their ride/driver. 5. The customer is ends up back on the home page and is able to request a new ride.
Use cases related to primary path: Leave a Review
Alternatives: 3.1 The customer directly navigates to the app without the notification and is still taken to review ride/driver screen. 4.1 The customer follows the primary path and also leaves a tip for the driver. 4.2 The customer doesn't leave a review and goes straight back to the home page.
Use cases related to alternatives: Leave a Review, Leave a Tip
Exceptions: The rider in question has notifications disabled on their device.
Use cases related to exceptions: None identified.
Notes: N/A

2.2 Requirements

1. The system **must** allow a rider to request a ride.

    1.1 A rider **must** be able to select a pickup point.

        1.1.1 A rider **must** be able to manually enter a postcode or location for their pickup point.

        1.1.2 A rider **should** be able to use their current location (calculated via GPS) as their pickup point.

    1.2 A rider **must** be able to specify a drop off point.

    1.3 A rider **must** be able to choose the type of vehicle it needs, e.g. number of seats, mobility support etc.

    1.4 A rider **must** be able to cancel a ride.

    1.5 A rider **should** receive push notifications on updates to their ride.

        1.5.1 A rider **should** receive a notification to leave a rating for their journey.

        1.5.2 A rider **should** receive a notification on ride accepted.

        1.5.3 A rider **should** receive a notification on ride cancelled.

        1.5.4 A rider **could** receive a notification on how far the driver is away.

    1.6 A rider **should** be able to see the route the driver is going to take to their destination.

    1.7 A rider **could** be able to alter the exact location of drop off and pickup points.

    1.8 A rider **could** be able to leave notes to help the driver find the pickup point.

2. The system **must** be able to take payment from a rider for a selected journey.

    2.1 The system **must** be able accept credit/debit payments.

    2.2 The system **should** be able to accept mobile payments from Google and Apple.

    2.3 The system **could** accept cryptocurrency payments, with consent from the driver.

    2.4 The system **won't** accept cash as a payment.

3. The system **must** allow both riders and drivers to sign up to the service.

    3.1 A customer (Rider/Driver) **must** be given a form to enter their:

        - name

- email
- phone number
- password
- banking details

3.1.1 A customer **must** be shown any inputs that fail validation.

3.1.2 A customers banking details **must** be valid.

3.1.3 A customers phone number **should** be validated using regex.

3.1.4 A customers email **should** be validated using regex.

3.1.5 A customers password **could** be checked for strength.

3.2 Uber **must** check a drivers background to align with regulations in that drivers area.

3.3 A customer (Rider/Driver) **must** be shown the terms and conditions.

3.3.1 A customer (Rider/Driver) **must** accept the terms and conditions.

3.3.2 A customer (Rider/Driver) **should** have to scroll through before accepting the terms and conditions.

3.4 A customer (Rider/Driver) **should** be notified on successful sign up.

3.5 A customer (Rider/Driver) **should** not have to sign in again on the device they successfully signed up on.

3.6 A customer (Rider/Driver) **could** be able to sign up using a third party (Google, Facebook etc.).

4. The system **must** offer safety features for customers (both rider and drivers).

4.1 The system **must** anonymise customers phone numbers to prevent communication after a ride.

4.2 The system **must** have a way for customers to alert authorities in the event of an incident.

- Police
- Fire Department
- Ambulance

4.3 Uber **must** run DBS and background checks on all drivers.

4.4 The system **should** allow riders to share their trip with friends and family.

4.5 The system **could** monitor a ride for deviations of route or expected travel time and contact the rider to make sure nothing has happened

3 Activity and Sequence Diagrams

Now that the use cases and requirements for Uber have been identified, I will now create activity and sequence diagrams to show how these high level use cases can be implemented into a system.

3.1 Activity Diagram

Below is an activity diagram that represents the 'Schedule Ride' use case previously identified. This diagram assumes that the user has already signed up to the app, so this is not included and does not model anything after the rider and driver have met.

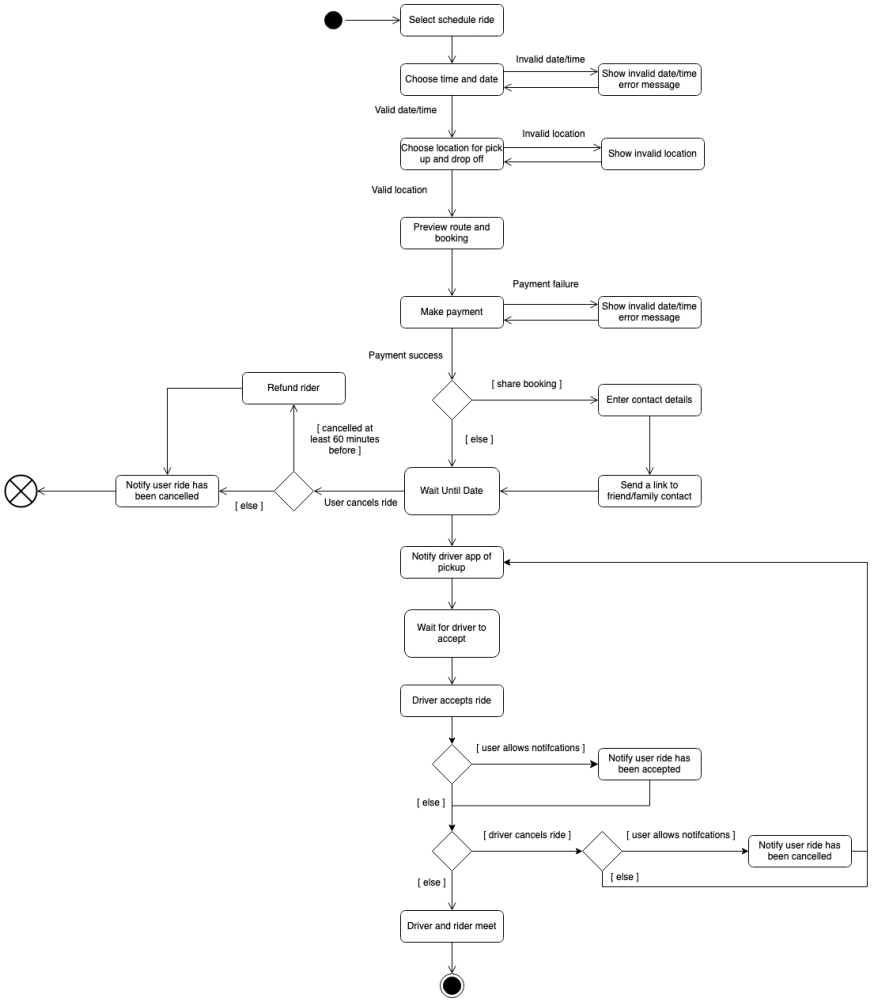

Figure 6: Figure showing activity diagram for Rider schedules a ride.

The rider must first select the schedule ride option within the app then they hit two activities in a row where they have to enter a date and drop off/pick up points, these activities both have *invalid* events that trigger error messages until the rider enters correct details. The booking/route can then be previewed and once again loops until the riders payment doesn't fail.

The rider is then given the decision to share their journey, otherwise they go straight to the 'Wait Until Date' state. Whilst in this state the user can cancel their ride and the system then makes a decision whether or not to refund the user, based on timeliness of cancellation. This is an exception to the happy/primary path.

After the scheduled date is hit, drivers are then sent notification of the ride and then once again hit some state this time waiting for a driver to accept the ride. Once a driver accepts a ride a decision is modeled to show whether or not the rider accepts notifications on their device, if they do a notification is sent, otherwise the flow continues. The driver then decides whether or not to cancel the ride, if they do the same notification logic is done as before, with a different message, and the drivers are once again notified of the available ride. If the driver does not cancel then the activity diagram concludes.

Where decisions are made represents the alternative paths available, a driver canceling a ride, a rider sharing a booking and whether or not the rider has notifications allowed on their device are all alternate paths to the main happy path.

3.2 Sequence Diagram

In addition to an activity diagram, I have also created a sequence diagram modeling the 'Confirm Ride Completion' use case for the rider actor. This use case happens once a driver has completed a ride and dropped a rider off at their selected drop off point. For this reason the diagram only models what happens after all a ride has finished. Signing up to the app, accepting a ride and any events that happen during the ride are not modeled.

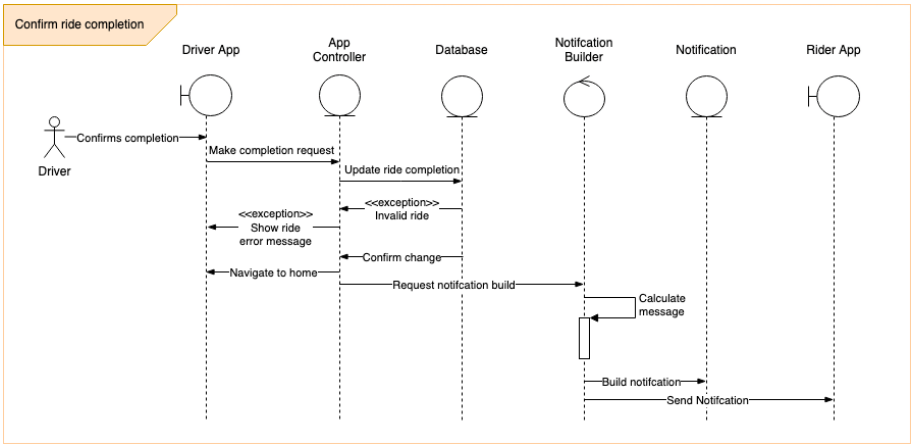

Figure 7: Figure showing sequence diagram for Driver confirming ride completion.

The driver uses the **Driver App** to confirm that a ride has completed. This interaction sends a request to the **App Controller** object which then updates the ride in **Database**. I chose do it this way as it's not a good idea to allow users of the app to have direct access to functions accessing database resources. The Database can throw an exception at this point if it is given a invalid ride to change. This results in the App Controller displaying an error message to the user. If a valid is given however the Database confirms the change with the App Controller and the driver is navigated back to the home screen.

In the background, with no additional input from the driver, the App Controller requests a notification to be built from the **Notification Builder** control object. This object calculates the notification message, builds a **Notification** object and then sends that Notification to the **Rider App**. I have included Notification as a separate object as there will be many different types of notifications, so this represents a base class which Notification Builder can tailor specific notifications.

Unlike in previous diagrams, here I am assuming the Rider App device is accepting notifications, otherwise there would be an exception here as the notification would never reach the Rider App.

4 Enterprise Architecture using Archimate

For documenting the enterprise architecture of Uber I have decided to use Archimate which:

> *'defines a common language for describing the purpose, construction, and operation of the organization.'* [4]

I have split this next section up into first modeling the business level, then the application level. For ease I have separated these levels into individual figures, however the fully connected diagram can be seen in the figure in **Appendix B**

4.1 Business Level

I decided to model the registration process for both riders and drivers, as this is a feature/functionality I haven't discussed much yet in this report. Starting with the business layer, this assumes that the user already has installed/downloaded the app to their device.

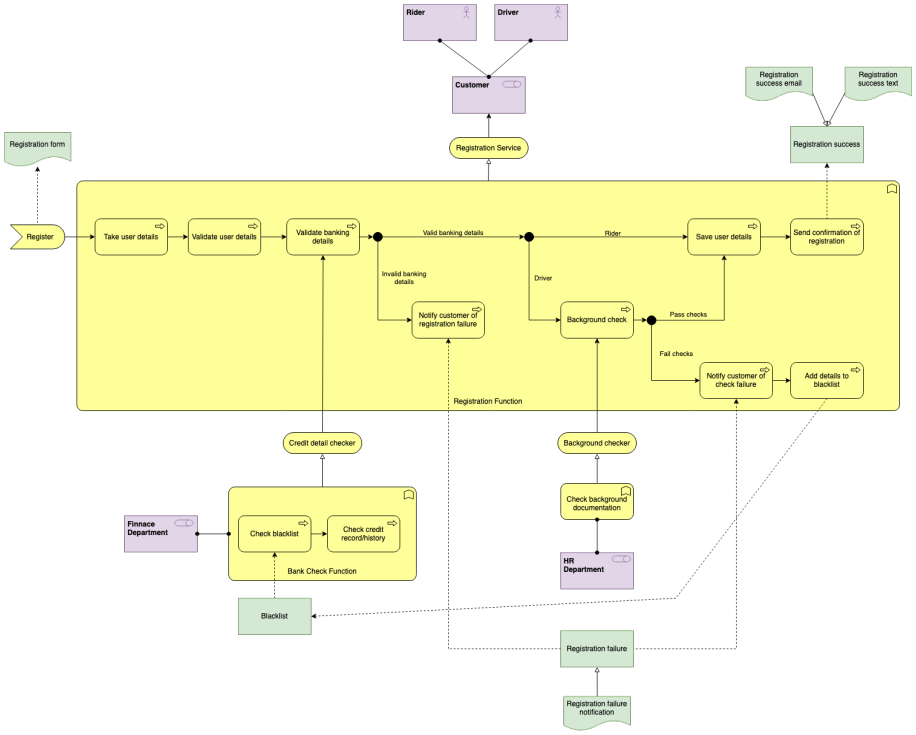

Figure 8: Figure showing archimate diagram showing business layer for registration process.

As this is for both rider and driver actors they are assigned the role of **Customer** which uses the **Registration Service** realised by the **Registration Function**. This is used to fire the **Register** event which starts the Registration Function. First Uber will take the customers specified account details and then validate them, then it will attempt to validate the customers banking details. This is done by the **Finance Department** using their Bank Check Function which checks if the given details are blacklisted, using the Blacklist data object as well as manually checking the users credit history.

If this fails the customer is sent a notified of registration failure. Otherwise the flow continues and checks whether or not this is a rider or driver attempting to sign up. If it's a driver background checks are done by the **HR Department** through the **Background Checker Service**. If this fails the user is notified again of registration failure and then added to the blacklist. I do modeled it this way as I am assuming Uber will not want anyone who has failed the background checks to ever be a driver for them.

If the driver passes the background check it rejoins the same path as the rider, finishing up the function by saving the new user details and sending the user confirmation of their successful sign up.

4.2 Application Level

For the application level I took functions/processes from the business level and thought whether or not they would need to be programmed into the system, instead of being done by an individual. From this analysis:

- Four services were modelled Email Service, Notification Service, Blacklist Service and Registration Service.
- Interfaces were modeled for these services with a 1-to-1 relationship, e.g. Blacklist Service used a Blacklist Interface. the only exception to this was the Notification Service which had two interfaces. This was due to the different types of notifications that the function was capable of producing.
- Two components were modeled, a Notification Component and a Registration Handler Component.

Notification Component - The Notification Component implements all interfaces provided by the Email Service and Notification Service. I grouped these together as the email service in this case is simply sending a success email, which is very similar to a push notification.

The component is assigned two functions, Send success email, which is realised by the Email Service and Send Notification, which is realised by the Notification Service and uses the Notification data object.

Registration Handler Component - The Registration Handler Component implements all interfaces provided by the Blacklist Service and Registration Service. Similar to the Notification Component I have grouped these services as blacklisting is part of the sign up process and triggers on attempted validation of a users registration details.

The Registration Handler Component is assigned three functions. The Add details to blacklist table uses the blacklist record data object, and is realised by the Blacklist Service. The Registration service realises two functions, the first being Validate registration details which also uses the Blacklist record object. The second is Save new user to user table which uses the User record data object.

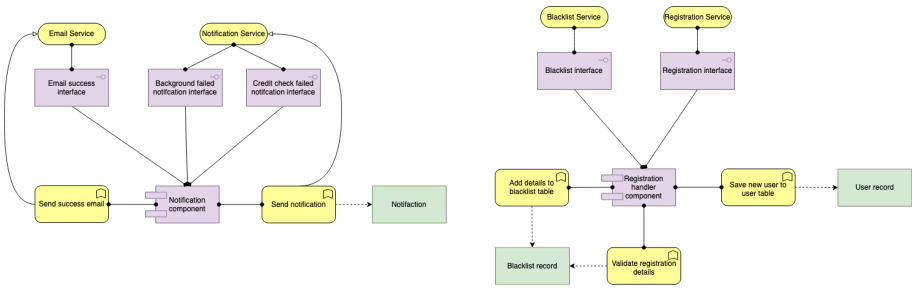

Figure 9: Figure showing archimate diagram showing application layer for registration process.

5 Implementing Self Driving Vehicles (Roadmap)

In my opinion a great long term goal for Uber is to implement self driving 'taxis' into the Uber eco-system. Tesla is the current leader in the self driving car, or CAV (Connected and Autonomous Vehicles), industry so will be the company/system that I suggest to integrate with. A few reasons why I think Tesla is the best option:

- As previously stated Tesla is currently the leader in this field.
- Uber already offers a service where users can rent a Tesla and gain benefits from doing so. [1]
- The above offer was due to a deal between Tesla and Hertz, so the company is clearly willing to negotiate deals with external companies.
- With Uber having the largest market share, (72%) over Lyfts' (28%) [2], in the ride sharing sector, it positions them well to get better and more profitable deals.

The implementation of a fully automated, self driving fleet of vehicles would mean that **riders** would end up with a more consistent/reliable service. This would result in a more loyal customer base, which in turn would result in more riders/rides and more profit. In addition to this, Uber would no longer have to take a cut of the drivers fare, they can take the whole fare for themselves as they would own the vehicles. Another profitability factor is the fact that there would be no need to continue support for the old driver app. This would mean money and time could be saved as:

1. Developers wouldn't have to work on the old driver app anymore, so development could go into new projects.
2. Old infrastructure, such as cloud or on premise servers, could be retired.
3. As previously mentioned Uber would receive the whole trip as payment, not just a cut.
4. Departments such as Customer Service could be cut down and have less issues to deal with, due to the total user base shrinking as drivers would no longer be a customer.

5.1 Motivation and Strategy Diagram

With Ubers already talented development team and their ability to hire new talent, due to their notoriety, I assume Uber has the technical skills to implement and make changes to their current infrastructure quickly. Below is an archimate strategy to demonstrate some of the points raised above.

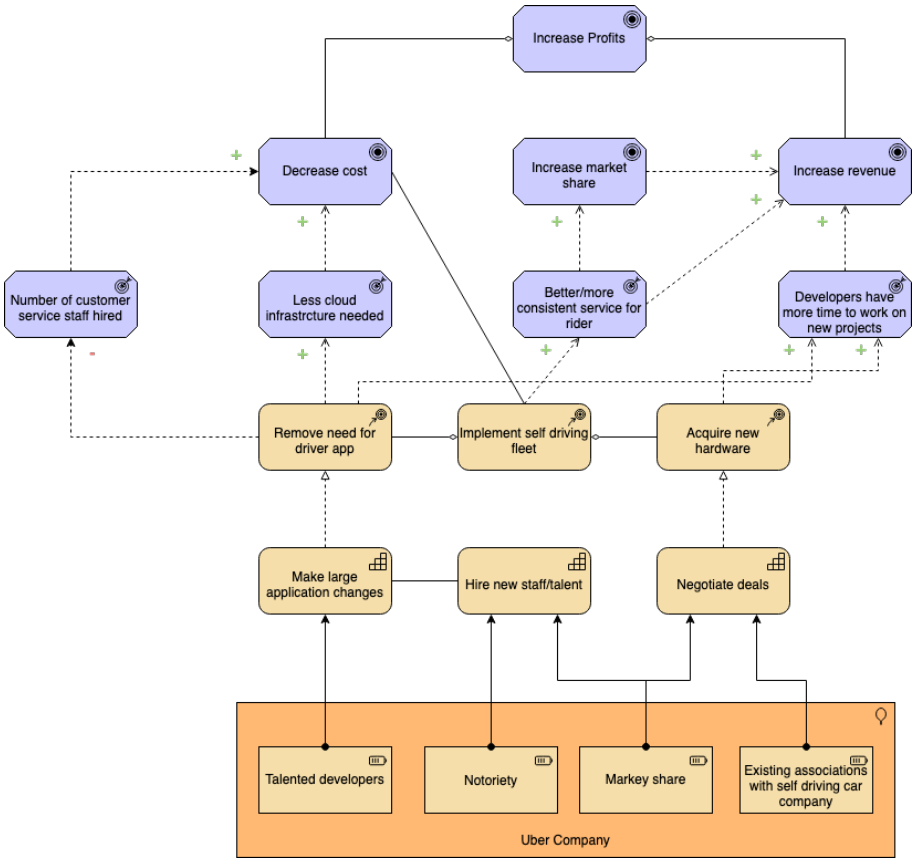

Figure 10: Figure showing archimate strategy diagram for implementation of self driving fleet.

Alongside the above strategy diagram I have also created a motivation diagram, again using archimate. This shares some of the same sentiment as the strategy diagram, however also dives more into customer satisfaction and the negative impact that can have on Uber.

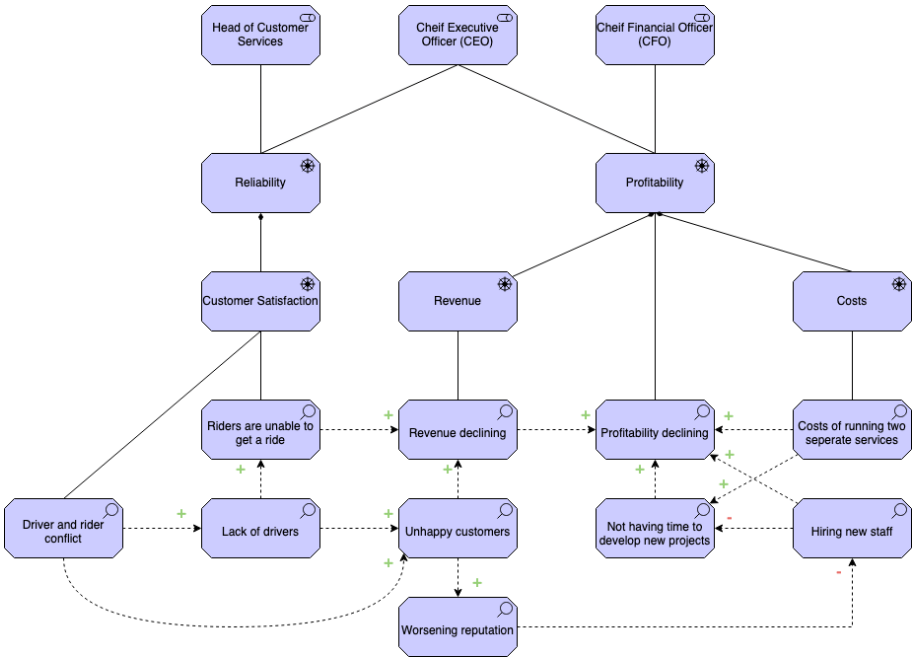

Figure 11: Figure showing motivation diagram showing current assessments and drivers of system.

Finally I have created implementation/migration diagram to show the stages of development for the suggested self driving fleet. This diagram is shown below and I will discuss it more on the next page.

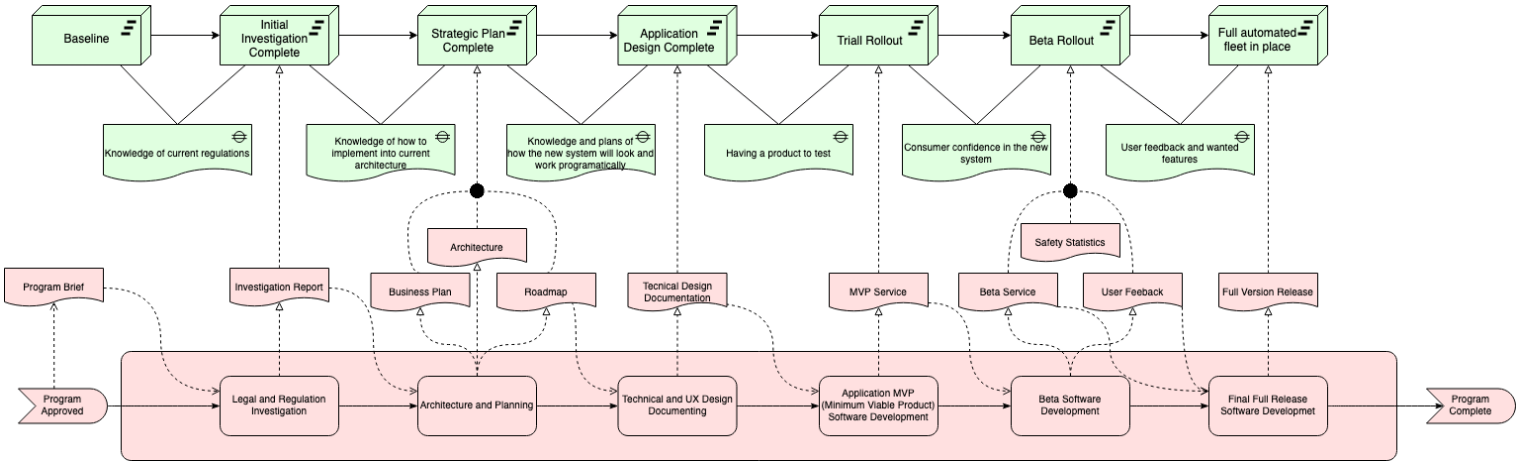

Figure 12: Figure showing archimate implementation/migration diagram.

6 Conclusion

7 References

[1] Uber. (2022) *The History of Uber - Uber's Timeline — Uber Newsroom.* Available at https://www.uber.com/en-GB/newsroom/history/ (accessed on 30th May 2022)

[2] Uber. (2022) *About us — Uber.* Available at https://www.uber.com/gb/en/about/ (accessed on 30th May 2022)

[3] Stripe Inc. (2022) *Online payment processing for internet businesses - Stripe.* Available at https://stripe.com/gb (accessed on 31st May 2022)

[4] The Open Group. (2022) *ArchiMate® Specification — The Open Group Website.* Available at https://www.opengroup.org/archimate-home (accessed on 31st May 2022)

[1] Uber. (2022) *Rent a Tesla & Drive with Uber — Uber.* Available at `https://www.uber.com/us/en/drive/vehicle-solutions/hertz/tesla/` (accessed on 27th May 2022).

[2] Perry, J. (2022) *The U.S. Rideshare Industry: Uber vs. Lyft - Bloomberg Second Measure.* Available at `https://secondmeasure.com/datapoints/rideshare-industry-overview/` (accessed on 27th May 2022).

8 Appendix

8.1 Appendix A : Full Use Case Diagram

Figure 13: Figure showing full Use Case Diagram

8.2 Appendix B : Full Archimate Registration Diagram

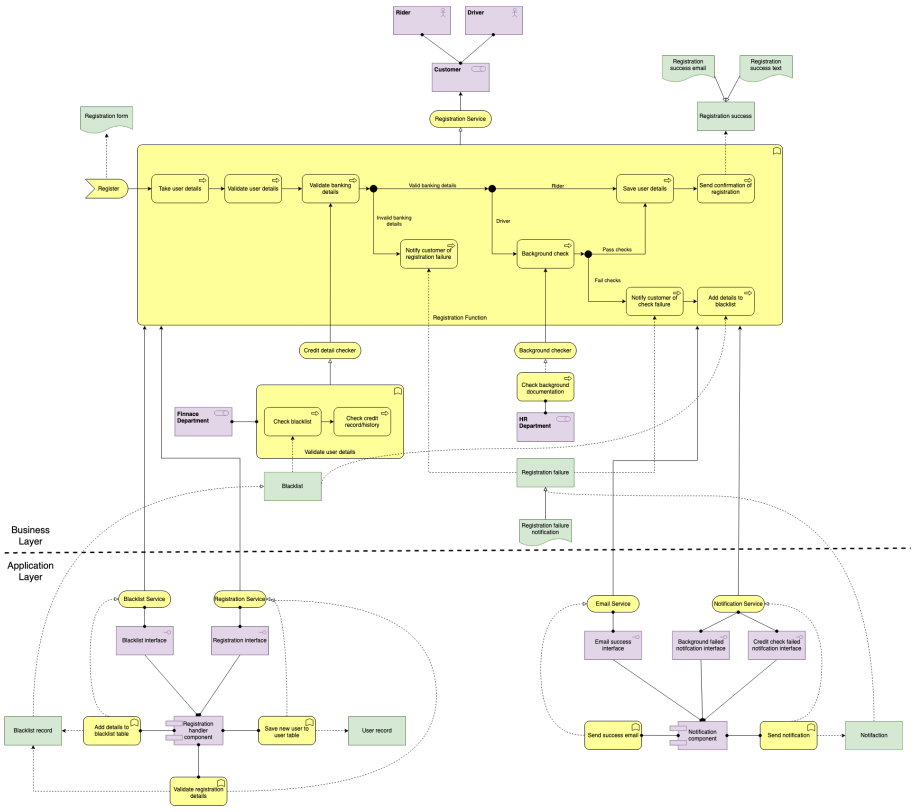

Figure 14: Figure showing archimate diagram showing process for registering an account.

8.3 Appendix C : Sequence Diagram with Access

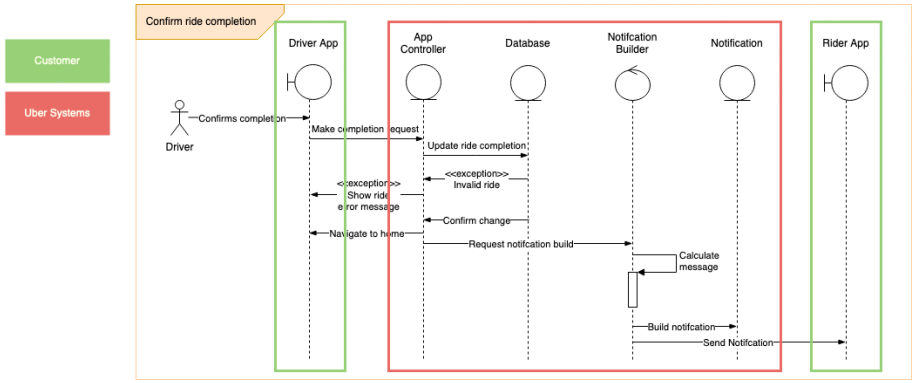

Figure 15: Figure showing sequence diagram showing access of objects.